Міністерство освіти і науки України

Національний університет «Києво-Могилянська академія»

Факультет інформатики

Кафедра математики

Кваліфікаційна робота

освітній ступінь – бакалавр

на тему: «**Оптимальні стратегії в багатокроковій грі зі скінченним горизонтом**»

Виконав: студент 4-го року навчання,

Спеціальності

113 Прикладна математика

Верещак Олексавндр Олегович

Керівник Чорней Р.К., _____

доцент, кандидат наук

Рецензент _____

Кваліфікаційна робота захищена

З оцінкою _____

Секретар ЕК _____

«____» _____20____р.

ЗМІСТ

АНОТАЦІЯ

У дипломній роботі досліджуються оптимальні стратегії в багатокрокових процесах прийняття рішень зі скінченним горизонтом. Розглядається математична модель Марковського процесу прийняття рішень (МППР) як інструмент для моделювання систем, що еволюціонують у часі під впливом керованих дій в умовах стохастичної невизначеності. Основна увага приділяється задачам зі скінченною кількістю етапів прийняття рішень.

У теоретичній частині роботи представлено формальний опис елементів МППР: станів, дій, функції винагороди та ймовірностей переходу. Детально проаналізовано рівняння Беллмана для скінченного горизонту як фундаментальне співвідношення для визначення оптимальної функції цінності. Описано алгоритм динамічного програмування, зокрема метод зворотної індукції, що дозволяє знаходити оптимальну нестаціонарну стратегію.

Практична частина роботи включає розробку програмного засобу мовою C# для реалізації описаного алгоритму. Продемонстровано можливості програми на прикладах знаходження оптимальних стратегій та відповідних функцій цінності для модельних задач. Також реалізовано функціонал симуляції процесу за знайденою оптимальною стратегією для оцінки фактично отриманої винагороди та порівняння з теоретичними очікуваннями. Проведено аналіз отриманих результатів, що підтверджують коректність роботи алгоритму та програмної реалізації.

Ключові слова: оптимальні стратегії, Марковський процес прийняття рішень, моделювання, керовані дії, рівняння Белмана, скінчений горизонт, нестаціонарна стратегія, симуляція.

ВСТУП

Сучасний світ характеризується складними системами та процесами, що розвиваються в умовах невизначеності та потребують прийняття послідовних рішень для досягнення бажаних результатів. Такі задачі виникають у різноманітних сферах, включаючи інженерію, економіку, робототехніку, управління ресурсами та логістику. Ефективне управління подібними системами вимагає розробки формальних моделей та алгоритмів, здатних знаходити найкращі послідовності дій з урахуванням стохастичної природи середовища та обмеженості часового горизонту.

Одним із потужних математичних інструментів для моделювання та аналізу таких задач є теорія Марковських процесів прийняття рішень. МППР дозволяють формалізувати взаємодію гравця із середовищем, де гравець на кожному кроці обирає дію, отримує винагороду та переходить до нового стану згідно з певними ймовірностями. Особливий інтерес становлять МППР зі скінченним горизонтом, де кількість етапів прийняття рішень заздалегідь обмежена. У таких моделях оптимальна стратегія, як правило, є нестаціонарною, тобто залежить від поточного кроку часу.

Актуальність даної дипломної роботи зумовлена необхідністю розробки та дослідження ефективних методів знаходження оптимальних стратегій для керованих стохастичних процесів зі скінченною тривалістю. Розуміння принципів побудови таких стратегій та наявність програмних засобів для їх розрахунку дозволяє підвищити ефективність прийняття рішень у багатьох практичних застосуваннях, де необхідно планувати дії наперед з урахуванням обмеженого часового ресурсу.

Метою даної дипломної роботи є дослідження методів знаходження оптимальних стратегій у Марковських процесах прийняття рішень зі скінченним горизонтом та розробка програмного засобу для їх реалізації та аналізу.

Для досягнення поставленої мети необхідно вирішити наступні **завдання**:

1. Проаналізувати теоретичні основи Марковських процесів прийняття рішень зі скінченним горизонтом, включаючи ключові поняття, рівняння Беллмана та методи їх розв'язання.

2. Детально розглянути алгоритм динамічного програмування (метод зворотної індукції) як основний інструмент для знаходження оптимальних стратегій у задачах зі скінченним горизонтом.

3. Розробити програмний модуль мовою C# для реалізації зазначеного алгоритму, що дозволяє обчислювати оптимальну функцію цінності та відповідну оптимальну стратегію.

4. Реалізувати функціонал симуляції процесу за знайденою оптимальною стратегією для емпіричної оцінки її ефективності.

5. Провести експериментальне дослідження роботи розробленого програмного засобу на модельних прикладах МППР, проаналізувати отримані результати та порівняти їх з теоретичними очікуваннями.

6. Сформулювати висновки щодо проведеної роботи та окреслити можливі напрямки подальших досліджень.

Об’єктом дослідження є багатокрокові процеси прийняття рішень в умовах стохастичної невизначеності з фіксованим, скінченним горизонтом планування.

Предметом дослідження є оптимальні нестаціонарні стратегії та алгоритми їх знаходження на основі методу динамічного програмування для Марковських процесів прийняття рішень зі скінченним горизонтом.

Застовуються методи теорії ймовірностей, теорія Марковських процесів, метод динамічного програмування, а також методи комп'ютерного моделювання та програмної реалізації алгоритмів.

РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ МАРКОВСЬКИХ ПРОЦЕСІВ ПРИЙНЯТТЯ РІШЕНЬ ЗІ СКІНЧЕННИМ ГОРИЗОНТОМ

1.1. Марковські процеси прийняття рішень як інструмент моделювання керованих стохастичних систем

У сучасному світі існує велика кількість складних систем, поведінка яких змінюється з часом під впливом як випадкових факторів, так і керованих дій. Задачі управління такими системами з метою досягнення певних цілей виникають у найрізноманітніших галузях: від автоматизованого керування технологічними процесами та робототехнічними комплексами до планування економічної діяльності, управління інвестиційними портфелями, оптимізації логістичних ланцюгів та навіть моделювання поведінки в біологічних системах. Ключовою особливістю таких задач є необхідність прийняття послідовних рішень в умовах невизначеності щодо майбутнього стану системи.

Для формального опису та аналізу подібних керованих стохастичних процесів широко застосовується математичний апарат теорії Марковських процесів прийняття рішень. Марковські процеси прийняття рішень надають гнучку та потужну структуру для моделювання ситуацій, де "гравець" (особа, що приймає рішення) взаємодіє з динамічним середовищем протягом послідовних часових етапів. На кожному етапі гравець спостерігає поточний стан середовища та обирає одну з доступних дій. Вибір дії призводить до отримання певної миттєвої винагороди (або штрафу) та переходу системи до нового стану, причому цей перехід визначається розподілом ймовірностей. Метою гравця є знаходження такої послідовності дій, або стратегії, яка максимізує деякий критерій ефективності, в нашому випадку, сумарну очікувану винагороду за певний період.

Марковські процеси прийняття рішень є узагальненням класичних Марковських ланцюгів. Якщо в Марковських ланцюгах ймовірності переходу між станами фіксовані та не залежать від зовнішнього впливу, то в Марковських процесах прийняття рішень ці переходи стають керованими: вибір дії гравцем безпосередньо впливає на розподіл ймовірностей переходу до наступних станів. Саме ця можливість активного впливу на динаміку системи через послідовність обраних дій робить Марковські процеси прийняття рішень потужним інструментом для розв'язання задач оптимального керування та планування. Серед них важливе місце займають процеси зі скінченним горизонтом, де кількість кроків є фіксованою.

1.2. Формальний опис моделі Марковського процесу прийняття рішень

Для того, щоб математично строго описати та проаналізувати процес прийняття послідовних рішень в умовах невизначеності, використовується модель Марковського процесу прийняття рішень (надалі МППР). Ця модель

структурує задачу, виділяючи її ключові елементи, що дозволяє системно підходити до пошуку оптимальної поведінки гравця. Класичний МППР визначається через наступні фундаментальні компоненти [1]:

1. **Множина станів (S):** Уявімо, що система, якою ми керуємо, може перебувати в різних конфігураціях або ситуаціях. Кожна така унікальна конфігурація є **станом** системи. Множина S – це повний перелік усіх можливих станів. Наприклад, для робота-пилососа станом може бути його місцезнаходження в кімнаті чи рівень заряду батареї. Позначимо стан системи в момент часу t як $s_t \in S$. Важливо, щоб стан містив усю релевантну інформацію, необхідну для прийняття обґрунтованого рішення саме в цей момент. У багатьох прикладних задачах ця множина є скінченною: S = {1, 2 ..., N}, де N – загальна кількість станів.
2. **Множина дій ($A(i)$):** Перебуваючи в певному стані i, гравець має можливість обрати одну з декількох доступних йому дій. Множина $A(i)$ якраз і визначає цей набір можливих вчинків для стану s. Наприклад, робот-пилосос у стані "низький заряд" може обрати дію "рухатись до зарядної станції" або "продовжити прибирання". Загальна множина всіх унікальних дій у системі позначається A. Дія, обрана гравцем в момент часу t, позначається $A_t \in A(S_t)$. Як правило, для кожного стану кількість доступних дій є скінченною.
3. **Функція ймовірностей переходу (P):** Цей компонент описує динаміку системи, тобто як вона змінює свій стан під впливом дій гравця та стохастичних факторів. Функція $p_{ij}(a) = P\{S_{t+1} = j \,|S_t = i,\, A_t = a\}$ визначає ймовірність того, що система перейде в стан j на наступному часовому етапі $(t + 1)$, якщо на поточному етапі вона перебувала в стані i і гравець виконав дію a. Наприклад, якщо робот обирає дію "рухатись вперед", існує ймовірність, що він успішно переміститься, але також є ймовірність зіткнутися з перешкодою і залишитися на місці

або навіть зміститися назад.

Ключовою особливістю МППР є **Марковська властивість**: майбутнє залежить лише від поточного стану та дії, а не минулого. Сума ймовірностей переходу в усі можливі наступні стани з будь-якого стану i при будь-якій дії a завжди дорівнює одиниці: $\sum_{s=1}^{N} p_{ij}(a) = 1$

4. **Функція винагороди (R):** Щоб гравець міг оцінювати ефективність своїх дій та прагнути до досягнення поставленої мети, вводиться поняття **винагороди**. Функція винагороди $r(i, a)$ визначає чисельну величину, яку гравець отримує негайно після виконання дії a в стані i.
5. **Часові етапи (t):** Прийняття рішень та еволюція системи відбуваються на дискретних часових етапах, які нумеруються t = 0, 1, 2, ... Кожен етап відповідає одному циклу: спостереження стану, вибір дії, отримання винагороди та перехід до нового стану. У задачах зі скінченним горизонтом існує заздалегідь визначений останній етап T, що обмежує тривалість процесу.

Ці п'ять компонентів $\{S, A, P, R, T\}$ є Марковським процесом прийняття рішень. На основі цієї моделі стає можливим формально поставити та розв'язати задачу знаходження оптимальної послідовності дій, що максимізує загальну очікувану винагороду для гравця.

1.3. Поняття стратегії в задачах зі скінченним горизонтом

Після визначення основних компонентів Марковського процесу прийняття рішень, наступним важливим кроком є формалізація поняття стратегії. Стратегія являє собою правило або набір правил, якими керується гравець при виборі дій на кожному етапі процесу з метою максимізації загальної очікуваної винагороди. Вибір оптимальної стратегії є центральною задачею в

теорії МППР.

Формально, стратегія π визначає для кожного можливого стану $i \in S$ та кожного етапу часу t правило вибору наступної дії $a \in A(i)$. Існує декілька способів класифікації стратегій [1]:

1. **Детерміновані та стохастичні (змішані) стратегії:**

Детермінована стратегія для кожного стану i та етапу t однозначно вказує одну конкретну дію a, яку слід виконати. Тобто, функція вибору дії повертає єдину дію.

Стохастична стратегія для кожного стану i та етапу t визначає розподіл ймовірностей над множиною доступних дій $A(i)$. Тобто, $f_t(i,a)$ є ймовірністю вибору дії a в стані i на етапі t, де $\sum_{a \in A(i)} f_t(i,a) = 1$.

Детермінована стратегія є окремим випадком стохастичної, де одна з ймовірностей дорівнює 1, а решта – 0. Для багатьох класів МППР, включаючи ті, що розглядаються в роботі, доведено, що існує оптимальна стратегія, яка є детермінованою [1].

2. **Стаціонарні та нестаціонарні стратегії:**

Стаціонарна стратегія π = f означає, що правило вибору дії f(i) залежить лише від поточного стану i і не змінюється з часом (тобто, $f_t(i) = f(i)$ для всіх t). Такі стратегії часто є оптимальними для задач з нескінченним горизонтом.

Нестаціонарна стратегія $\pi = (f_0, f_1, \ldots, f_{\{T-1\}})$ (або f_T, залежно від визначення горизонту) означає, що правило вибору дії $f_t(i)$ може бути різним для різних етапів часу t, навіть якщо система перебуває в тому самому стані i.

У задачах Марковських процесів прийняття рішень зі скінченним горизонтом, які є предметом даного дослідження, оптимальна стратегія, як правило, є нестаціонарною. Це інтуїтивно зрозуміло: рішення, яке є найкращим, коли до кінця процесу залишилося багато етапів, може бути неоптимальним, коли залишилося лише кілька етапів. Наприклад, на початку тривалого процесу гравець може дозволити собі ризиковані дії з потенційно

високою винагородою в майбутньому, тоді як наприкінці горизонту він, ймовірно, обере більш безпечні дії, що гарантують негайний, хоч і менший, результат. Таким чином, оптимальна стратегія для скінченного горизонту представляється як послідовність функцій вибору дій $\pi^* = (f_0^*, f_1^* \dots, f_{\{T-1\}}^*)$, де кожна f_t^*визначає оптимальну дію для кожного стану на відповідному етапі t.

1.4. Критерій оптимальності та функція цінності в МППР на скінченному горизонті

Після визначення основних компонентів Марковського процесу прийняття рішень та поняття стратегії, необхідно формалізувати критерій, за яким оцінюватиметься ефективність тієї чи іншої стратегії. У задачах МППР зі скінченним горизонтом таким критерієм зазвичай виступає максимізація сумарної очікуваної винагороди, отриманої гравцем протягом усього горизонту подій T. Нехай $S_0 = i$ - початковий стан системи, а $\pi = (f_0, f_1 \dots, f_{\{T-1\}})$ - деяка допустима нестаціонарна стратегія. Тоді цільовою функцією, яку необхідно максимізувати, є загальна очікувана винагорода, що може бути представлена у вигляді формули [1]: $V_T(i, \pi) = \sum_{t=0}^{T} E_{i\pi}[R_t]$. Для аналізу та знаходження оптимальної стратегії вводяться поняття функції цінності стану та, що тісно з нею пов'язана, функції цінності стану-дії.

Функція цінності стану для стратегії π: Функція цінності $V_\pi(i, t)$ визначає очікувану сумарну винагороду, яку отримає гравець, починаючи з етапу t у стані i і дотримуючись стратегії π до кінця горизонту T: $V_\pi(i, t) = \sum_t^T E_\pi[R_t|S_n = i]$. Ця функція дозволяє оцінити "якість" перебування в певному стані і на певному етапі t при використанні обраної стратегії π.

Оптимальна функція цінності стану: Оптимальна функція цінності $V^*(i, t)$

представляє максимальну можливу очікувану сумарну винагороду, яку можна отримати, починаючи з етапу t у стані i і діючи оптимально до кінця горизонту: $V^*(i,t) = max_\pi(V_\pi(i,t))$. У термінології зворотної індукції, де n позначає кількість кроків, що залишилися до кінця, оптимальна функція цінності $V_n^*(i)$ визначає максимальну очікувану винагороду за ці n кроків, починаючи зі стану i, при цьому $V_0^*(i) = 0$.

Функція цінності стану-дії для стратегії π: Поряд з функцією цінності стану, важливою є також функція цінності стану-дії $Q_\pi(i,a,t)$. Вона визначає очікувану сумарну винагороду, якщо на етапі t в стані i гравець виконає конкретну дію a, а потім, починаючи з наступного етапу, буде дотримуватися стратегії π [2]: $Q_\pi(i,a,t) = r(i,a) + E_\pi[\sum_{k=t+1}^{T-1} r(S_k, A_k) \,|S_t, A_t = a]$, де перший член $r(i,a)$ – це миттєва винагорода, а другий – очікувана сумарна винагорода за всі наступні етапи при дотриманні стратегії π після виконання дії a у стані i.

Оптимальна функція цінності стану-дії: Аналогічно до V^*, оптимальна функція цінності стану-дії $Q^*(i,a,t)$ (або $Q_n^*(i,a)$ в термінах n кроків до кінця) представляє максимальну очікувану сумарну винагороду, якщо на поточному етапі в стані і виконати дію a, а потім дотримуватися оптимальної стратегії [2] π^*: $Q_n^*(i,a) = r\,(i,a) + \sum p_{ij}(a) * V_{n-1}^*(j)$. Ця формула показує, що оптимальна цінність виконання дії a в стані i (коли залишилося n кроків) складається з негайної винагороди $r(i,a)$ та очікуваної оптимальної цінності стану $V_{n-1}^*(j)$, до якого система перейде, зваженої на ймовірність цього переходу $p_{ij}(a)$ (математичне сподівання).

Зв'язок між V^* та Q^*: Оптимальна функція цінності стану $V_n^*(i)$ пов'язана з оптимальною функцією цінності стану-дії $Q_n^*(i,a)$ наступним чином: $V_n^*(i) = max_a(Q_n^*(i,a))$. Це означає, що оптимальна цінність стану i (коли залишилося n кроків) досягається шляхом вибору такої дії a, яка максимізує $Q_n^*(i,a)$. Знання оптимальних функцій цінності (V^* та Q^*) дозволяє безпосередньо визначити оптимальну стратегію π^*. Оптимальна стратегія –

це така стратегія, яка для кожного стану i та кожного етапу t (або для кожної кількості кроків до кінця n) обирає дію, що веде до досягнення $V^*(i,t)$, тобто максимізує $Q^*(i,a,t)$. Як буде написано далі, рівняння Беллмана надають рекурсивний спосіб обчислення цих оптимальних функцій та відповідної оптимальної стратегії.

1.5. Рівняння Беллмана та метод динамічного програмування

Центральним математичним апаратом для знаходження оптимальних стратегій у Марковських процесах прийняття рішень зі скінченним горизонтом є рівняння Беллмана, названі на честь Річарда Беллмана, який сформулював принцип оптимальності. Ці рівняння надають рекурсивний спосіб визначення оптимальної функції цінності [3].

Як було зазначено раніше, оптимальна функція цінності $V_n^*(i)$ представляє максимальну очікувану сумарну винагороду, яку можна отримати, починаючи зі стану i, коли до кінця процесу залишилося n етапів. Рівняння Беллмана для скінченного горизонту має наступний вигляд[1]:

$$V_n^*(i) = max_{Ai}\{r(i,a) + \sum_{i=1}^{N} p_{ij}(a) * V_{n-1}^*(j)\}$$

де n змінюється від 1 до T. Ініціалізація відбувається умовою $V_0^*(i)$= 0 для всіх $i \in S$, що означає відсутність майбутніх винагород після завершення всіх етапів. Дане рівняння стверджує, що оптимальна цінність стану i за n етапів до кінця дорівнює максимальному значенню, яке можна отримати, обравши деяку дію a зі множини доступних дій $A(i)$. Це значення складається з негайної винагороди $r(i,a)$ за поточну дію та очікуваної оптимальної цінності $V_{n-1}^*(j)$ наступного стану j, помноженої на ймовірність переходу $p_{ij}(a)$.

Для практичного розв'язання рівнянь Беллмана та знаходження оптимальної стратегії використовується метод динамічного програмування, а саме його

варіант, відомий як алгоритм зворотної індукції. Цей алгоритм безпосередньо реалізує рекурсивне співвідношення Беллмана, працюючи "назад у часі", кроки роботи алгоритму:

Ініціалізація: На самому кінці горизонту, коли не залишилося жодного етапу для прийняття рішень (n=0), встановлюється $V_0^*(i)$= 0 для всіх станів $i \in S$.

Ітераційний крок: Починаючи з n = 1 і рухаючись до n = T (тобто від передостаннього етапу до початкового), для кожного стану $i \in S$ виконуються наступні обчислення:

а) Знаходиться дія a^*, яка максимізує вираз:

$a^*(i,n) = argmax_{Ai} \{r(i,a) + \sum_{j=1}^{N} p_{ij}(a) * V_{n-1}^*(j)\}$

б) Обчислюється оптимальна функція цінності для поточного стану і та n етапів до кінця:

$$V_n^*(i) = max_{Ai} \{r(i,a) + \sum_{i=1}^{N} p_{ij}(a) * V_{n-1}^*(j)\}$$

Результат: Після виконання всіх ітерацій (до n=T), ми отримуємо:

Оптимальну функцію цінності $V_n^*(i)$ для всіх станів i та всіх можливих кількостей етапів до кінця n. Зокрема, $V_T^*(i)$ представляє максимальну очікувану сумарну винагороду за весь процес, якщо він починається зі стану i.

Оптимальну нестаціонарну стратегію $\pi^* = (f_0^*, f_1^* \dots, f_{T-1}^*)$де кожна функція $f_t^*(i)$ вказує оптимальну дію в стані і на етапі часу t.

Таким чином, метод динамічного програмування надає спосіб розв'язання задачі оптимального керування для Марковських процесів прийняття рішень зі скінченним горизонтом.

РОЗДІЛ 2. РОЗРОБКА ПРОГРАМНОГО ЗАСОБУ ДЛЯ ПОШУКУ ОПТИМАЛЬНИХ СТРАТЕГІЙ В МППР ЗІ СКІНЧЕННИМ ГОРИЗОНТОМ

2.1 Архітектура програмного модуля та представлення даних

Розробка програмного забезпечення для розв'язання задач Марковських процесів прийняття рішень зі скінченним горизонтом вимагає ретельного проектування архітектури та вибору адекватних структур даних для ефективного представлення моделі та результатів обчислень. Для реалізації було обрано мову програмування C#, що завдяки своїм об'єктно-орієнтованим властивостям та розвиненій стандартній бібліотеці є зручним інструментом для подібних завдань.

Основна логіка програмного модуля інкапсульована в спеціально створеному класі FiniteHorizonMDPSolver. Цей клас призначений для зберігання всіх параметрів МППР, виконання обчислень за алгоритмом динамічного програмування, проведення симуляцій та виведення отриманих результатів. Внутрішня структура класу організована таким чином, щоб чітко розділити вхідні дані, проміжні обчислення, кінцеві результати та їх вивід у консоль.

Представлення вхідних параметрів моделі МППР:

Для зберігання визначальних характеристик Марковського процесу прийняття рішень у класі FiniteHorizonMDPSolver використовуються наступні поля та типи даних:

Кількість станів (numStates): Зберігається як цілочисельна змінна типу int. Визначає загальну кількість унікальних станів N у множині S = {0, 1, ..., N-1}. Використання цілочисельнього типу є природним для підрахунку станів.

Максимальна кількість дій (maxActions): Також цілочисельна змінна типу

int. Визначає верхню межу для кількості дій, доступних у будь-якому стані. Це полегшує визначення розмірності масивів, хоча фактична кількість дій $|A(i)|$ може бути меншою для конкретного стану i.

Горизонт (horizonT): Цілочисельна змінна int, що позначає загальну кількість етапів прийняття рішень T. Етапи часу t нумеруються від 0 до T-1.

Матриця ймовірностей переходу (transitionProbs): Представлена тривимірним масивом типу double[,,]. Елемент transitionProbs[i, a, j] зберігає ймовірність $p_{ij}(a)$ переходу зі стану i в стан j при виконанні дії a. Тип double обрано для представлення ймовірностей, які є дійсними числами в діапазоні [0, 1], з достатньою точністю. Тривимірна структура масиву забезпечує прямий доступ до потрібної ймовірності за індексами стану, дії та наступного стану.

Функція винагороди (rewards): Реалізована як двовимірний масив double[,]. Елемент rewards[i, a] містить значення миттєвої винагороди $r(i, a)$, отримуваної гравцем при виконанні дії a в стані i. Використання типу double дозволяє представляти як позитивні, так і негативні винагороди (штрафи) з необхідною точністю.

Структури даних для зберігання результатів обчислень:

Результати роботи алгоритму динамічного програмування зберігаються у наступних полях класу:

Оптимальна функція цінності (valueFunction): Двовимірний масив double[,] розмірністю [T+1, N]. Елемент valueFunction[n, i] зберігає значення оптимальної функції цінності $V_n^*(i)$, тобто максимальну очікувану сумарну винагороду, якщо система перебуває в стані i та до кінця горизонту залишилося n етапів (де n змінюється від 0 до T). Перший індекс n відповідає кількості кроків до кінця, що є звичним для алгоритму зворотної індукції.

Оптимальна стратегія (optimalPolicy): Двовимірний масив int[,] розмірністю [T, N]. Елемент optimalPolicy[t, i] зберігає індекс оптимальної дії $f_t^*(i)$, яку

слід виконати на етапі часу t (від 0 до T-1), перебуваючи в стані i. Тип int використовується для зберігання індексів дій.

Вибір багатовимірних масивів для представлення цих ключових елементів моделі та результатів обґрунтований їх ефективністю для доступу до даних за індексами, що безпосередньо відповідає математичній нотації та логіці алгоритму динамічного програмування. Для проведення симуляцій також використовується поле randomGenerator типу System.Random для генерації випадкових чисел, необхідних для моделювання стохастичних переходів. Така архітектура та вибір структур даних забезпечують логічну організацію програмного модуля та створюють основу для ефективної реалізації обчислювальних функцій.

2.2. Алгоритмічна реалізація методу динамічного програмування

Центральним елементом розробленого програмного засобу є реалізація алгоритму динамічного програмування методом зворотної індукції, який обчислює оптимальну функцію цінності та відповідну оптимальну стратегію. Ця логіка реалізована в основному методі класу FiniteHorizonMDPSolver, названому Solve(). Робота методу базується на ітеративному розв'язанні рівняння Беллмана рухаючись від кінцевого етапу горизонту до початкового. Процес обчислень в методі Solve() можна розділити на наступні ключові етапи:

Ініціалізація функції цінності:

На початку роботи алгоритму відбувається ініціалізація значень оптимальної функції цінності для моменту часу, коли до кінця горизонту не залишилося жодного етапу прийняття рішень (тобто, n=0 кроків до кінця). Для всіх станів $i \in S$ встановлюється valueFunction[0, i] = 0.0. Це значення відповідає нульовій майбутній винагороді після завершення всіх етапів процесу і слугує базовим випадком для рекурсивних обчислень за рівнянням Беллмана.

Цикл зворотної індукції:

Основні обчислення відбуваються в циклі, який ітерує по кількості етапів n, що залишилися до кінця горизонту, від n=1 до T (де T – загальна кількість етапів, що відповідає значенню horizonT у програмній реалізації). На кожній ітерації цього циклу (для кожного n):

Визначається відповідний поточний етап часу t = T - n. Цей індекс t використовується для запису знайденої оптимальної дії в масив стратегій.

Для кожного стану $i \in S$ системи (від 0 до numStates-1) виконується пошук оптимальної дії та обчислення оптимальної цінності:

1. Ініціалізуються допоміжні змінні: maxQValue (для зберігання максимального знайденого Q-значення для поточної пари (i, n)) початковим значенням double.NegativeInfinity, та bestAction (для зберігання індексу відповідної найкращої дії) значенням -1 (що означає " дія не визначена").

2. Здійснюється перебір усіх можливих дій a (від 0 до maxActions-1), які гравець може виконати в поточному стані i. Перед обчисленням викликається метод IsActionValid(int state, int action) для перевірки, чи є поточна дія a допустимою для стану i (тобто, чи визначені для неї коректні ймовірності переходу). Якщо дія невалідна, вона пропускається.

3. Для кожної валідної дії a:

3.1. З масиву rewards[i, a] отримується значення миттєвої винагороди $r(i, a)$.

3.2. Обчислюється очікувана майбутня цінність expectedFutureValue. Це досягається шляхом ітерації по всіх можливих наступних станах $j \in S$ (від 0 до numStates-1) та підсумовування добутків ймовірностей переходу transitionProbs[i, a, j] (що відповідає $p_{ij}(a)$) на значення оптимальної функції цінності valueFunction[n-1, j] (тобто $V_{n-1}^{*}(j)$), яке було розраховане на попередній ітерації по n (коли до кінця залишався n-1 етап):

expectedFutureValue = $\sum_{j=0}^{N-1}(\text{transitionProbs[i, a, j]} * \text{valueFunction[n} - 1, \text{j]})$

3.3. Розраховується повне Q-значення (цінність пари стан-дія) для поточної

пари (i, a) та n кроків до кінця:
currentQValue = rewards[i, a] + expectedFutureValue
3.4. Якщо обчислене currentQValue перевищує поточне значення maxQValue, то maxQValue оновлюється значенням currentQValue, а bestAction – індексом поточної дії a.
4. Після перебору всіх доступних дій для стану i: якщо була знайдена хоча б одна валідна дія (bestAction != -1), то знайдене maxQValue присвоюється елементу valueFunction[n, i], що відповідає значенню $V_n^*(i)$. Знайдений індекс bestAction зберігається в optimalPolicy[t, i], що відповідає оптимальній дії $f_t^*(i)$.
Після завершення всіх ітерацій зовнішнього циклу по n, масиви valueFunction та optimalPolicy міститимуть, відповідно, повну оптимальну функцію цінності для всіх станів та всіх можливих кількостей етапів до кінця горизонту, та повну оптимальну нестаціонарну стратегію для всіх станів та всіх етапів часу від 0 до T-1.

2.3. Реалізація модуля симуляції процесу

Окрім безпосереднього обчислення оптимальної стратегії та функції цінності, важливою складовою розробленого програмного засобу є модуль симуляції. Симуляція дозволяє продемонструвати поведінку гравця, який дотримується знайденої оптимальної стратегії, в умовах стохастичних переходів, а також провести оцінку ефективності цієї стратегії шляхом порівняння фактично отриманих винагород з теоретичними очікуваннями. У класі FiniteHorizonMDPSolver за реалізацію симуляції відповідають методи SimulateOnce(int startState) та RunSimulations(int numSimulationsPerStartState). Метод SimulateOnce(int startState) призначений для проведення одного повного циклу Марковського процесу прийняття рішень, починаючи з заданого початкового стану startState та тривалістю T етапів (від t=0 до T-1).

Логіка роботи методу наступна:

Ініціалізація симуляції:

Поточний стан системи currentState встановлюється рівним startState.

Змінна totalSimulatedReward для накопичення загальної отриманої винагороди ініціалізується нулем.

Для наочності процесу симуляції відбувається виведення інформації про початковий стан.

Цикл по етапах часу:

Здійснюється ітерація по всіх етапах часу t від 0 до T-1. На кожному етапі:

1. Вибір дії за оптимальною стратегією: Визначається оптимальна дія actionToTake, яку слід виконати в поточному стані currentState на поточному етапі t. Ця дія береться з попередньо розрахованого масиву optimalPolicy[t, currentState].

2. Обробка непередбачених ситуацій: Якщо для поточної пари (стан, етап) оптимальна дія не була визначена (наприклад, actionToTake == -1), симуляція буде перервана, з відповідним повідомленням.

3. Накопичення миттєвої винагороди: До загальної винагороди totalSimulatedReward додається миттєва винагорода rewards[currentState, actionToTake], отримана за виконання обраної дії в поточному стані.

4. Симуляція стохастичного переходу: Для визначення наступного стану системи моделюється випадковий перехід:

4.1. Генерується випадкове число randomRoll з рівномірного розподілу на інтервалі [0, 1) за допомогою об'єкта System.Random.

4.2. Ітеративно перебираються всі можливі наступні стані $j \in S$. Накопичується сума ймовірностей переходу cumulativeProb до цих станів, використовуючи значення transitionProbs[currentState, actionToTake, j].

4.3. Коли randomRoll стає меншим за поточне значення cumulativeProb, стан j обирається як наступний стан nextState, і цикл перебору наступних станів переривається. Цей механізм реалізує вибір наступного стану згідно з заданим розподілом ймовірностей.

4.4 Оновлення стану системи: Поточний стан currentState оновлюється на nextState.

4.5 Відображення кроків: Для аналізу інформації виводиться виконана дія та стан, до якого відбувся перехід.

Завершення симуляції: Після проходження всіх T етапів, виводиться загальна отримана винагорода totalSimulatedReward за дану симуляцію. Метод повертає це значення.

Метод RunSimulations(int numSimulationsPerStartState) використовується для проведення серії симуляцій та узагальнення результатів:

Метод приймає параметр numSimulationsPerStartState, що визначає, скільки разів потрібно запустити симуляцію для кожного можливого початкового стану.

Для кожного стану $i \in S$, який розглядається як початковий:

Викликається метод SimulateOnce(int i) вказану кількість разів.

Підсумовуються отримані загальні винагороди.

Обчислюється середня фактична винагорода для даного початкового стану за всі проведені симуляції.

Ця середня винагорода виводиться разом із теоретично розрахованою оптимальною очікуваною винагородою valueFunction[T, i] для порівняння.

Використання модуля симуляції дозволяє отримати практичне уявлення про те, як система поводиться при дотриманні оптимальної стратегії, та наскільки близькими є результати окремих реалізацій до середніх очікуваних значень.

РОЗДІЛ 3. ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ТА АНАЛІЗ РЕЗУЛЬТАТІВ

3.1. Приклад

У даному розділі представлено застосування розробленого програмного засобу, що реалізує алгоритм динамічного програмування для Марковських процесів прийняття рішень зі скінченним горизонтом, до низки модельних задач. Метою експериментального дослідження є демонстрація працездатності алгоритму, верифікація програмної реалізації через порівняння з розрахунками для простого прикладу, а також аналіз результатів симуляційного моделювання.

Для початкової ілюстрації роботи алгоритму та верифікації програмного коду розглянемо просту задачу вибору шляху, де гравець має дістатися фінішної точки, обираючи між більш безпечними, але менш винагороджуваними діями, та більш ризикованими, але потенційно більш вигідними

3.1.1. Постановка задачі та визначення параметрів моделі МППР

Визначимо компоненти Марковського процесу прийняття рішень для даного прикладу:
Горизонт планування: T = 2.
Множина станів: N=3
Стан 0: Старт
Стан 1: Проміжний вузол
Стан 2: Фініш
Множина дій: A=2
Дія 0: Обережний рух
Дія 1: Ризикований рух

Функція винагороди:

Таблиця 3.1 Функція винагороди R(i,a)

i \ a	0	1
0	-1	-3
1	2	6
2	5	5

Ймовірності переходу:

Таблиця 3.2

Ймовірності переходу для дії 0 (Обережний рух) зі стану i в стан j:

i \ j	0	1	2
0	0	1	0
1	0.1	0.1	0.8
2	0	0	1

Таблиця 3.3

Ймовірності переходу для дії 1 (Ризикований рух) зі стану i в стан j:

i \ j	0	1	2
0	0.3	0.3	0.4
1	0.4	0.1	0.5
2	0	0	1

Очікується, що оптимальна стратегія буде залежати від етапу: на останньому етапі гравець, ймовірно, обере дію, що максимізує негайну винагороду, тоді як на першому етапі він має враховувати потенційні майбутні винагороди від ризикованіших дій.

3.1.2. Покроковий розрахунок оптимальної функції цінності та стратегії для задачі

Застосуємо алгоритм зворотної індукції для знаходження оптимальної функції цінності $V_n^*(i)$ та оптимальної стратегії π^*.

1. Ініціалізація (n = 0):

Згідно з алгоритмом, цінність станів після завершення всіх етапів дорівнює нулю:

$V_0^*(0) = 0$

$V_0^*(1) = 0$

$V_0^*(2) = 0$

2. Розрахунок для n = 1:

На цьому етапі обчислюємо $V_1^*(i) = \max\{\mathrm{r}(i,a) + \sum_{j=0}^{2} p_{ij}(a) * V_0^*(j)\}$.

Оскільки $V_0^*(j) = 0$ для всіх j, то формула спрощується до $V_1^*(i) = \max\{\mathrm{r}(i,a)\}$.

Оптимальна дія $f_1^*(i)$ буде та, яка дає максимальну миттєву винагороду:

Для стану i = 0:

Дія 0: $Q_1 = r(0,0) = -1$

Дія 1: $Q_1 = r(0,1) = -3$

$V_1^*(0) = \max(-1,-3) = -1$

Отже, оптимальна дія $f_1^*(0) = 0$.

Для стану i = 1:

Дія 0: $Q_1 = r(1,0) = 2$

Дія 1: $Q_1 = r(1,1) = 6$

$V_1^*(1) = \max(2,6) = 6$

Отже, оптимальна дія $f_1^*(1) = 1$.

Для стану i = 2:

Дія 0: $Q_1 = r(2,0) = 5$

Дія 1: $Q_1 = r(2,1) = 5$

$V_1^*(2) = \max(5,5) = 5$

Отже, оптимальна дія $f_1^*(2) = 0$ (при рівності, нехай, обираємо 0).

Проміжні результати для n=1:

$V_1^*(0) = -1,\ f_1^*(0) = 0$

$V_1^*(1) = 6,\ f_1^*(1) = 1$

$V_1^*(2) = 5,\ f_1^*(2) = 0$

3. Розрахунок для n = 2:

Обчислимо $V_2^*(i) = \max\{\mathrm{r}(i,a) + \sum_{j=0}^{2} p_{ij}(a) * V_1^*(j)\}$.

Для стану i = 0 :

Дія 0: $Q_2(0,0) = r(0,0) + p_{00}(0) * V_1^*(0) + p_{01}(0) * V_1^*(1) + p_{02}(0) * V_1^*(2) = -1 - 1 * 0 + 1 * 6 + 5 * 0 = 5$

Дія 1: $Q_2(0,1) = r(0,1) + p_{00}(1) * V_1^*(0) + p_{01}(1) * V_1^*(1) + p_{02}(1) * V_1^*(2) = -3 + 0.3 * (-1) + 0.3 * 6 + 0.4 * 5 = 0.5$

$V_2^*(0) = \max(5, 0.5) = 5$

Отже, оптимальна дія $f_0^*(0) = 0$.

Для стану i = 1:

Дія 0: $Q_2(1,0) = r(1,0) + p_{10}(0) * V_1^*(0) + p_{11}(0) * V_1^*(1) + p_{12}(0) * V_1^*(2) = 2 - 1 * 0.1 + 0.1 * 6 + 0.8 * 5 = 6.5$

Дія 1: $Q_2(1,1) = r(1,1) + p_{10}(1) * V_1^*(0) + p_{11}(1) * V_1^*(1) + p_{12}(1) * V_1^*(2) = 5 - 1 * 0.4 + 0.1 * 6 + 0.6 * 5 = 8.7$

$V_2^*(1) = \max(6.5, 8.7) = 8.7$

Отже, оптимальна дія $f_0^*(1) = 1$.

Для стану i = 2:

Дія 0: $Q_2(2,0) = r(2,0) + p_{20}(0) * V_1^*(0) + p_{21}(0) * V_1^*(1) + p_{22}(0) * V_1^*(2) = 5 - 1 * 0 + 6 * 0 + 1 * 5 = 10$

Дія 1: $Q_2(2,1) = r(2,1) + p_{20}(1) * V_1^*(0) + p_{21}(1) * V_1^*(1) + p_{22}(1) * V_1^*(2) = 5 - 1 * 0 + 6 * 0 + 1 * 5 = 10$

$V_2^*(2) = \max(10, 10) = 10$

Отже, оптимальна дія $f_0^*(2) = 0$.

Підсумкові таблиці

Таблиця 3.4

Оптимальна Функція Цінності $V_n^*(i)$:

n \ i	0	1	2
0	0	0	0
1	-1	6	5
2	5	8.7	10

Таблиця 3.5

Оптимальна стратегія $\pi^* = (f_0^*, f_1^*)$:

t \ i	0	1	2
0	0	1	0
1	0	1	0

3.1.3. Результати роботи програмного засобу для задачі та їх порівняння з розрахунком

Для верифікації коректності розробленого програмного засобу, передамо початкові умови задачі вибору шляху у метод Main() класу FiniteHorizonMDPSolver.

```
public static void Main(string[] args)
{
    //  Початкові параметри
    int nStates = 3;
    int mActions = 2;
    int T = 2;

    // Масиви для збереження обрахованих даних
    double[,,] P = new double[nStates, mActions, nStates];
    double[,] R = new double[nStates, mActions];

    // Винагороди R[стан, дія]
    R[0, 0] = -1;
    R[0, 1] = -3;
    R[1, 0] = 2;
    R[1, 1] = 6;
    R[2, 0] = 5;
    R[2, 1] = 5;

    // Ймовірності переходу P[поточний_стан, дія, наступний_стан]
    // Стан 0:
    P[0, 0, 0] = 0.0; P[0, 0, 1] = 1.0; P[0, 0, 2] = 0.0; // Дія 0
    P[0, 1, 0] = 0.3; P[0, 1, 1] = 0.3; P[0, 1, 2] = 0.4; // Дія 1

    // Стан 1:
    P[1, 0, 0] = 0.1; P[1, 0, 1] = 0.1; P[1, 0, 2] = 0.8; // Дія 0
    P[1, 1, 0] = 0.4; P[1, 1, 1] = 0.1; P[1, 1, 2] = 0.5; // Дія 1

    // Стан 2:
    P[2, 0, 0] = 0.0; P[2, 0, 1] = 0.0; P[2, 0, 2] = 1.0; // Дія 0
    P[2, 1, 0] = 0.0; P[2, 1, 1] = 0.0; P[2, 1, 2] = 1.0; // Дія 1
```

Рисунок 3.1 Початкові параметри програми

Після запуску програми, методом Solve() були обчислені оптимальна функція цінності та оптимальна стратегія. Результати роботи програми:

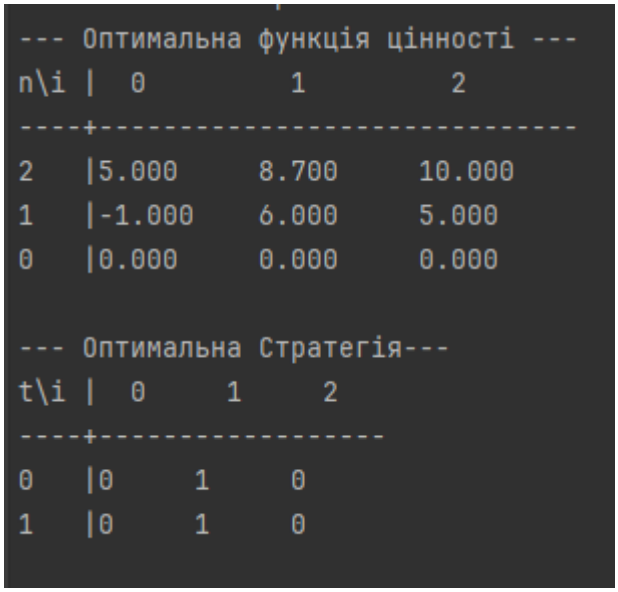

Рисунок 3.2 Результати роботи програми

Як ми можемо бачити, значення оптимальної функції цінності й оптимальної стратегії збігаються, що підтверджує коректність роботи алгоритму.

3.2 Робота програмного засобу на складніших задачах

Перейдемо до застосування програми на більш комплексній задачі. Це дозволить продемонструвати можливості розробленого алгоритму та провести детальний аналіз отриманих оптимальних стратегій та результатів симуляцій.

3.2.1. Приклад 2. Робот-кур’єр

Розглянемо задачу оптимального управління рухом робота-кур'єра, який має доставити вантаж з початкової точки до кінцевої за обмежену кількість кроків. Ключовим аспектом задачі є необхідність балансувати між швидкістю доставки та ризиком повного розряду батареї робота до завершення.

Визначення компонентів:

Горизонт планування: T = 3.

Множина станів: N = 5 станів, що описують комбінацію місцезнаходження робота та рівня заряду його батареї:

Стан 0: Початкова точка, Високий рівень заряду

Стан 1: На шляху до клієнта, Середній рівень заряду

Стан 2: На шляху до клієнта, Низький рівень заряду

Стан 3: Клієнт, Будь-який рівень заряду

Стан 4: Батарея розряджена

Множина дій: Для кожного стану доступні A = 2 дії:

Дія 0: Рухатись помірно

Дія 1: Рухатись швидко

Функція винагороди:

Таблиця 3.6

Функція винагороди R(i,a)

i \ a	0	1
0	-1	-3
1	-1	-3
2	-1	-3
3	100	100
4	-50	-50

Ймовірності переходу:

Таблиця 3.7

Ймовірності переходу для дії 0 зі стану i в стан j:

i \ j	0	1	2	3	4
0	0.1	0.8	0.1	0.0	0.0
1	0.0	0.2	0.6	0.2	0.0
2	0.0	0.0	0.3	0.4	0.3
3	0.0	0.0	0.0	1.0	0.0
4	0.0	0.0	0.0	0.0	1.0

Таблиця 3.8

Ймовірності переходу для дії 1 зі стану i в стан j:

i \ j	0	1	2	3	4
0	0.0	0.5	0.3	0.2	0.0
1	0.0	0.1	0.3	0.5	0.1
2	0.0	0.0	0.1	0.3	0.6
3	0.0	0.0	0.0	1.0	0.0
4	0.0	0.0	0.0	0.0	1.0

Ці ймовірності відображають компроміс між швидкістю досягнення цілі та витратою енергії. Наприклад, зі стану 0 при дії 1 існує 20% ймовірність одразу дістатися клієнта, але ймовірність перейти до станів із середнім або низьким зарядом $p_{01}(1) + p_{02}(1) = 0.8$ більша, ніж при повільному русі. Зі стану 2 швидкий рух має високу ймовірність призвести до повного розряду

батареї $p_{24}(1) = 0.6$.

Очікується, що оптимальна стратегія буде динамічно змінювати вибір дій. Наприклад, при високому рівні заряду та достатній кількості часу до кінця горизонту робот може обирати швидкий рух. Однак, при низькому рівні заряду або наближенні до кінця часового горизонту оптимальною може стати дія "Рухатись помірно". Також очікується, що стратегія буде уникати дій, що з високою ймовірністю ведуть до стану 4 (Батарея розряджена).

3.2.2. Аналіз результатів задачі управління роботом-кур'єром

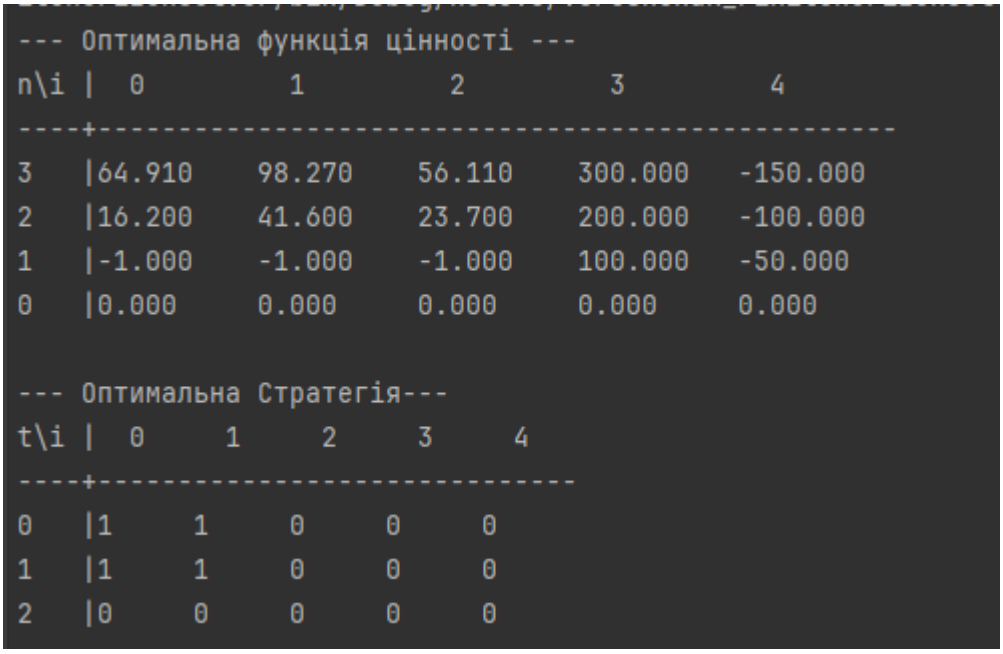

Рисунок 3.3 Результати роботи програми робота-кур'єра

Аналіз отриманої оптимальної стратегії: Проаналізуємо оптимальну стратегію, отриману програмним засобом для задачі управління роботом-кур'єром, розглядаючи вибір дій на кожному етапі та в кожному стані. Оптимальна стратегія є нестаціонарною, тобто вибір дії залежить не лише від стану, а й від етапу часу, що залишився до кінця горизонту.

На початковому етапі програма рекомендує діяти "Швидко" (дія 1) для станів 0 ("Початкова точка, Високий заряд") та 1 ("В дорозі, Середній заряд").

Незважаючи на більші негайні витрати енергії (штраф -3 проти -1 за помірний рух), високий початковий або середній заряд батареї дозволяють роботові ризикнути заради швидшого прогресу до цілі. Очікувана цінність майбутніх станів компенсує поточні витрати. Для стану 2 ("В дорозі, Низький заряд") програма обирає дію "Помірно" (дія 0). При низькому заряді ризик повного розряду при "швидкому" русі стає занадто високим, тому оптимально діяти обережніше, щоб максимізувати шанс досягти клієнта, незважаючи на потенційну повільність.

На середньому етапі оптимальні дії для станів 0, 1, 2, 3, 4 залишаються такими ж, як і на етапі t = 0. Це свідчить про те, що на цих проміжних етапах співвідношення ризик/винагорода для "Швидкого" та "Помірного" руху залишається стабільним, а робот ще має достатньо часу для маневрів.

На останньому етапі t = 2: Стратегія суттєво змінюється для станів 0, 1 та 2. Програма рекомендує діяти "Помірно" (дія 0) для всіх нетермінальних станів (0, 1, 2). Це логічно: якщо залишився лише один крок, а майбутня винагорода після нього нульова ($V_0^*(j) = 0$), гравець просто обирає дію, яка дає найкращу негайну винагороду. Для всіх цих станів дія "Помірно" має негайну винагороду -1, тоді як "Швидко" - -3, що робить "Помірно" кращим вибором. Для термінальних станів 3 та 4 стратегія залишається незмінною.

Загалом, отримана стратегія відповідає інтуїтивним очікуванням: робот дозволяє собі ризикувати на початку гри з високим зарядом, але стає обережнішим по мірі зменшення заряду або наближення кінця горизонту. Ця поведінка підкреслює адаптивний характер оптимальних стратегій у Марковських процесах прийняття рішень на скінченному горизонті.

Результати симуляційного моделювання та їх аналіз: було проведено симуляцію за допомогою функції симуляції. Для кожного можливого початкового стану було запущено по 10 симуляцій. Нижче наведено приклади траєкторій та порівняння фактично отриманих винагород з теоретично очікуваними.

```
--- Симуляції, що починаються зі стану: 0 ---
Симуляція зі стану 0: [0 -(дія 1)-> 1 -(дія 1)-> 3 -(дія 0)-> 3] -> Загальна винагорода: 94.000
Симуляція зі стану 0: [0 -(дія 1)-> 1 -(дія 1)-> 4 -(дія 0)-> 4] -> Загальна винагорода: -56.000
Симуляція зі стану 0: [0 -(дія 1)-> 1 -(дія 1)-> 4 -(дія 0)-> 4] -> Загальна винагорода: -56.000
Симуляція зі стану 0: [0 -(дія 1)-> 1 -(дія 1)-> 3 -(дія 0)-> 3] -> Загальна винагорода: 94.000
Симуляція зі стану 0: [0 -(дія 1)-> 1 -(дія 1)-> 3 -(дія 0)-> 3] -> Загальна винагорода: 94.000
Симуляція зі стану 0: [0 -(дія 1)-> 2 -(дія 0)-> 3 -(дія 0)-> 3] -> Загальна винагорода: 96.000
Симуляція зі стану 0: [0 -(дія 1)-> 3 -(дія 0)-> 3 -(дія 0)-> 3] -> Загальна винагорода: 197.000
Симуляція зі стану 0: [0 -(дія 1)-> 1 -(дія 1)-> 3 -(дія 0)-> 3] -> Загальна винагорода: 94.000
Симуляція зі стану 0: [0 -(дія 1)-> 2 -(дія 0)-> 3 -(дія 0)-> 3] -> Загальна винагорода: 96.000
Симуляція зі стану 0: [0 -(дія 1)-> 1 -(дія 1)-> 1 -(дія 0)-> 3] -> Загальна винагорода: -7.000
Середня винагорода для стартового стану 0 за 10 симуляцій: 64.600
Очікувана винагорода (V*[3,0]): 64.910
```

Рисунок 3.4 Результат симуляції починаючи зі стану 0

```
--- Симуляції, що починаються зі стану: 1 ---
Симуляція зі стану 1: [1 -(дія 1)-> 3 -(дія 0)-> 3 -(дія 0)-> 3] -> Загальна винагорода: 197.000
Симуляція зі стану 1: [1 -(дія 1)-> 3 -(дія 0)-> 3 -(дія 0)-> 3] -> Загальна винагорода: 197.000
Симуляція зі стану 1: [1 -(дія 1)-> 3 -(дія 0)-> 3 -(дія 0)-> 3] -> Загальна винагорода: 197.000
Симуляція зі стану 1: [1 -(дія 1)-> 2 -(дія 0)-> 2 -(дія 0)-> 2] -> Загальна винагорода: -5.000
Симуляція зі стану 1: [1 -(дія 1)-> 3 -(дія 0)-> 3 -(дія 0)-> 3] -> Загальна винагорода: 197.000
Симуляція зі стану 1: [1 -(дія 1)-> 2 -(дія 0)-> 2 -(дія 0)-> 3] -> Загальна винагорода: -5.000
Симуляція зі стану 1: [1 -(дія 1)-> 3 -(дія 0)-> 3 -(дія 0)-> 3] -> Загальна винагорода: 197.000
Симуляція зі стану 1: [1 -(дія 1)-> 1 -(дія 1)-> 3 -(дія 0)-> 3] -> Загальна винагорода: 94.000
Симуляція зі стану 1: [1 -(дія 1)-> 3 -(дія 0)-> 3 -(дія 0)-> 3] -> Загальна винагорода: 197.000
Симуляція зі стану 1: [1 -(дія 1)-> 3 -(дія 0)-> 3 -(дія 0)-> 3] -> Загальна винагорода: 197.000
Середня винагорода для стартового стану 1 за 10 симуляцій: 146.300
Очікувана винагорода (V*[3,1]): 98.270

--- Симуляції, що починаються зі стану: 2 ---
Симуляція зі стану 2: [2 -(дія 0)-> 2 -(дія 0)-> 2 -(дія 0)-> 4] -> Загальна винагорода: -3.000
Симуляція зі стану 2: [2 -(дія 0)-> 4 -(дія 0)-> 4 -(дія 0)-> 4] -> Загальна винагорода: -101.000
Симуляція зі стану 2: [2 -(дія 0)-> 3 -(дія 0)-> 3 -(дія 0)-> 3] -> Загальна винагорода: 199.000
Симуляція зі стану 2: [2 -(дія 0)-> 4 -(дія 0)-> 4 -(дія 0)-> 4] -> Загальна винагорода: -101.000
Симуляція зі стану 2: [2 -(дія 0)-> 4 -(дія 0)-> 4 -(дія 0)-> 4] -> Загальна винагорода: -101.000
Симуляція зі стану 2: [2 -(дія 0)-> 4 -(дія 0)-> 4 -(дія 0)-> 4] -> Загальна винагорода: -101.000
Симуляція зі стану 2: [2 -(дія 0)-> 3 -(дія 0)-> 3 -(дія 0)-> 3] -> Загальна винагорода: 199.000
Симуляція зі стану 2: [2 -(дія 0)-> 4 -(дія 0)-> 4 -(дія 0)-> 4] -> Загальна винагорода: -101.000
Симуляція зі стану 2: [2 -(дія 0)-> 4 -(дія 0)-> 4 -(дія 0)-> 4] -> Загальна винагорода: -101.000
Симуляція зі стану 2: [2 -(дія 0)-> 2 -(дія 0)-> 3 -(дія 0)-> 3] -> Загальна винагорода: 98.000
Середня винагорода для стартового стану 2 за 10 симуляцій: -11.300
Очікувана винагорода (V*[3,2]): 56.110
```

Рисунок 3.5 Результат симуляції починаючи зі стану 1 та 2

```
--- Симуляції, що починаються зі стану: 3 ---
Симуляція зі стану 3: [3 -(дія 0)-> 3 -(дія 0)-> 3 -(дія 0)-> 3] -> Загальна винагорода: 300.000
Симуляція зі стану 3: [3 -(дія 0)-> 3 -(дія 0)-> 3 -(дія 0)-> 3] -> Загальна винагорода: 300.000
Симуляція зі стану 3: [3 -(дія 0)-> 3 -(дія 0)-> 3 -(дія 0)-> 3] -> Загальна винагорода: 300.000
Симуляція зі стану 3: [3 -(дія 0)-> 3 -(дія 0)-> 3 -(дія 0)-> 3] -> Загальна винагорода: 300.000
Симуляція зі стану 3: [3 -(дія 0)-> 3 -(дія 0)-> 3 -(дія 0)-> 3] -> Загальна винагорода: 300.000
Симуляція зі стану 3: [3 -(дія 0)-> 3 -(дія 0)-> 3 -(дія 0)-> 3] -> Загальна винагорода: 300.000
Симуляція зі стану 3: [3 -(дія 0)-> 3 -(дія 0)-> 3 -(дія 0)-> 3] -> Загальна винагорода: 300.000
Симуляція зі стану 3: [3 -(дія 0)-> 3 -(дія 0)-> 3 -(дія 0)-> 3] -> Загальна винагорода: 300.000
Симуляція зі стану 3: [3 -(дія 0)-> 3 -(дія 0)-> 3 -(дія 0)-> 3] -> Загальна винагорода: 300.000
Симуляція зі стану 3: [3 -(дія 0)-> 3 -(дія 0)-> 3 -(дія 0)-> 3] -> Загальна винагорода: 300.000
Середня винагорода для стартового стану 3 за 10 симуляцій: 300.000
Очікувана винагорода (V*[3,3]): 300.000

--- Симуляції, що починаються зі стану: 4 ---
Симуляція зі стану 4: [4 -(дія 0)-> 4 -(дія 0)-> 4 -(дія 0)-> 4] -> Загальна винагорода: -150.000
Симуляція зі стану 4: [4 -(дія 0)-> 4 -(дія 0)-> 4 -(дія 0)-> 4] -> Загальна винагорода: -150.000
Симуляція зі стану 4: [4 -(дія 0)-> 4 -(дія 0)-> 4 -(дія 0)-> 4] -> Загальна винагорода: -150.000
Симуляція зі стану 4: [4 -(дія 0)-> 4 -(дія 0)-> 4 -(дія 0)-> 4] -> Загальна винагорода: -150.000
Симуляція зі стану 4: [4 -(дія 0)-> 4 -(дія 0)-> 4 -(дія 0)-> 4] -> Загальна винагорода: -150.000
Симуляція зі стану 4: [4 -(дія 0)-> 4 -(дія 0)-> 4 -(дія 0)-> 4] -> Загальна винагорода: -150.000
Симуляція зі стану 4: [4 -(дія 0)-> 4 -(дія 0)-> 4 -(дія 0)-> 4] -> Загальна винагорода: -150.000
Симуляція зі стану 4: [4 -(дія 0)-> 4 -(дія 0)-> 4 -(дія 0)-> 4] -> Загальна винагорода: -150.000
Симуляція зі стану 4: [4 -(дія 0)-> 4 -(дія 0)-> 4 -(дія 0)-> 4] -> Загальна винагорода: -150.000
Симуляція зі стану 4: [4 -(дія 0)-> 4 -(дія 0)-> 4 -(дія 0)-> 4] -> Загальна винагорода: -150.000
Середня винагорода для стартового стану 4 за 10 симуляцій: -150.000
Очікувана винагорода (V*[3,4]): -150.000
```

Рисунок 3.6 Результат симуляції починаючи зі стану 3 та 4

Спостерігається значна варіативність отриманих сумарних винагород для окремих ітерацій процесу, що починаються з одних і тих самих станів. Це є прямим наслідком випадкових переходів між станами. Наприклад, зі стану 0, який має теоретичну очікувану винагороду 64.910, спостерігалися результати від -7.000 до 197.000. При цьому середня винагорода за 10 симуляцій (64.600) для стану 0 є дуже близькою до очікуваного значення (64.910), що свідчить про коректність знайденої політики та модуля симуляції.

Водночас, для станів 1 та 2 спостерігаються більші відхилення середніх винагород за 10 симуляцій від теоретичних очікувань (наприклад, 146.300 проти 98.270 для стану 1, та -11.300 проти 56.110 для стану 2).

Для термінальних станів 3 та 4, де ймовірності переходу 1.0, середня винагорода за симуляції повністю збігається з теоретичною очікуваною

винагородою, що є додатковим підтвердженням коректної роботи. Це ще раз ілюструє, що оптимальна стратегія забезпечує максимально можливу очікувану винагороду в середньому, а не гарантує максимальний результат у кожному конкретному випадку через наявність випадкових факторів.

ВИСНОВКИ

У ході виконання даної дипломної роботи було досліджено оптимальні стратегії в Марковських процесах прийняття рішень зі скінченним горизонтом. Робота охоплювала як теоретичні аспекти моделювання, так і практичну реалізацію алгоритмів пошуку таких стратегій. Під час дослідження було здійснено декілька ключових кроків, найголовніші з яких:
Досліджено теоретичні основи Марковських процесів прийняття рішень зі скінченним горизонтом. Було детально проаналізовано математичну модель МППР, її основні компоненти (стани, дії, ймовірності переходу, функція винагороди), а також концепцію стратегії, з акцентом на нестаціонарних стратегіях, що є оптимальними для скінченного горизонту.
Застосовано Рівняння Беллмана та метод динамічного програмування (зворотну індукцію). Рівняння Беллмана було розглянуто як фундаментальне рекурсивне співвідношення для знаходження оптимальної функції цінності, а алгоритм зворотної індукції – як ефективний обчислювальний метод для його розв’язання.
Розроблено програмний засіб на мові C# для реалізації алгоритму динамічного програмування. Була визначена архітектура програмного модуля, спроектовані структури даних для представлення моделі МППР та її параметрів. Реалізовано основний метод Solve(), що обчислює оптимальну функцію цінності та стратегію, а також модуль симуляції для емпіричної перевірки поведінки системи.
Проведено експериментальне дослідження та аналіз оптимальних стратегій на модельних прикладах. На тестовому прикладі було проведено покроковий

ручний розрахунок та підтверджено коректність програмної реалізації. На складнішому прикладі було детально проаналізовано динамічну поведінку оптимальної стратегії залежно від стану та етапу часу, а також проведено симуляцію поведінки, що підтвердило наближення фактично отриманих винагород до теоретичних очікувань.

Дана робота демонструє можливості застосування теоретичних основ Марковських процесів прийняття рішень для розв’язання задач оптимального керування в умовах стохастичної невизначеності та обмеженого часового горизонту. Розроблений програмний засіб є дієвим інструментом для знаходження адаптивних стратегій та їх аналізу, що може слугувати основою для підтримки прийняття рішень у різноманітних прикладних галузях.

Напрямки подальших досліджень:

В подальшому можливо розширити функціональність розробленого програмного засобу та дослідити більш складні аспекти.

Це може включати:

Розгляд МППР з частково спостережуваними станами.

Впровадження дисконтованого критерію оптимальності.

Адаптація алгоритму для задач з нескінченним горизонтом.

Розробка графічного інтерфейсу користувача для зручнішого введення даних та візуалізації результатів

ДЖЕРЕЛА

1. Pascal J. Stochastic Games. Mathematisch Instituut, Universiteit Leiden 2006. P 1–36.

2. Filar J., Vrieze K. Markov Decision Processes: The Noncompetitive Case. Competitive Markov Decision Processes. New York, NY, 1997. P. 9–84.

3. Bellman equation [Електронний ресурс]. – 2024. – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Bellman_equation

ДОДАТОК

```
using System;
using System.Globalization;
using System.Linq;

public class FiniteHorizonMDPSolver
{
    private int numStates;
    private int maxActions;
    private int horizonT;
    private double[,,] transitionProbs;
    private double[,] rewards;

    private double[,] valueFunction;
    private int[,] optimalPolicy;

    private Random randomGenerator;

    public FiniteHorizonMDPSolver(int nStates, int mActions, int T, double[,,] P,
```

```
double[,] R)
   {
      if (nStates <= 0 || mActions <= 0 || T < 0)
         throw new ArgumentException("Кількість станів, дій і горизонт мають бути позитивними (T >= 0).");
      if (P.GetLength(0) != nStates || P.GetLength(1) != mActions || P.GetLength(2) != nStates)
         throw new ArgumentException("Неправильний розмір матриці ймовірностей переходу P.");
      if (R.GetLength(0) != nStates || R.GetLength(1) != mActions)
         throw new ArgumentException("Неправильний розмір матриці винагород R.");

      numStates = nStates;
      maxActions = mActions;
      horizonT = T;
      transitionProbs = P;
      rewards = R;

      valueFunction = new double[T + 1, numStates];
      optimalPolicy = new int[T, numStates];

      for (int tLoop = 0; tLoop < T; tLoop++)
      {
         for (int i = 0; i < numStates; i++)
         {
            optimalPolicy[tLoop, i] = -1;
         }
      }
      randomGenerator = new Random();
```

```
    }

    private bool IsActionValid(int state, int action)
    {
        if (action < 0 || action >= maxActions) return false;
        double probSum = 0;
        for (int nextState = 0; nextState < numStates; nextState++)
        {
            if (!double.IsNaN(transitionProbs[state, action, nextState]) &&
!double.IsInfinity(transitionProbs[state, action, nextState]))
            {
                probSum += transitionProbs[state, action, nextState];
            }
        }
        return Math.Abs(probSum - 1) < 0.001;
    }

    public (double[,] ValueFunction, int[,] OptimalPolicy) Solve()
    {
        for (int i = 0; i < numStates; i++)
        {
            valueFunction[0, i] = 0.0;
        }

        for (int n = 1; n <= horizonT; n++)
        {
            int t = horizonT - n;
            for (int i = 0; i < numStates; i++)
            {
                double maxQValue = double.NegativeInfinity;
```

```
            int bestAction = -1;
            for (int a = 0; a < maxActions; a++)
            {
               if (!IsActionValid(i, a))
               {
                  continue;
               }
               double immediateReward = rewards[i, a];
               double expectedFutureValue = 0;
               for (int j = 0; j < numStates; j++)
               {
                  expectedFutureValue += transitionProbs[i, a, j] * valueFunction[n -
1, j];
               }
               double currentQValue = immediateReward + expectedFutureValue;
               if (currentQValue > maxQValue)
               {
                  maxQValue = currentQValue;
                  bestAction = a;
               }
            }
            valueFunction[n, i] = maxQValue;
            optimalPolicy[t, i] = bestAction;
         }
      }
      return (valueFunction, optimalPolicy);
   }

   public void PrintResults()
   {
```

```
        Console.WriteLine("--- Оптимальна функція цінності ---");
        Console.Write("n\\i |");
        for (int i = 0; i < numStates; i++) Console.Write($"  {i,-8}");
        Console.WriteLine("\n----+" + new string('-', numStates * 10));
        for (int n = horizonT; n >= 0; n--)
        {
            Console.Write($"{n,-4}|");
            for (int i = 0; i < numStates; i++)
            {
                Console.Write($"{valueFunction[n, i].ToString("F3",
CultureInfo.InvariantCulture),-10}");
            }
            Console.WriteLine();
        }

        Console.WriteLine("\n--- Оптимальна Стратегія---");
        Console.Write("t\\i |");
        for (int i = 0; i < numStates; i++) Console.Write($"  {i,-4}");
        Console.WriteLine("\n----+" + new string('-', numStates * 6));
        for (int t = 0; t < horizonT; t++)
        {
            Console.Write($"{t,-4}|");
            for (int i = 0; i < numStates; i++)
            {
                Console.Write($"{optimalPolicy[t, i],-6}");
            }
            Console.WriteLine();
        }
        Console.WriteLine("(Значення = -1 - дія не визначена)");
```

```
        Console.WriteLine("\n---Очікувана винагорода за весь горизонт T ---");
        for (int i = 0; i < numStates; i++)
        {
            Console.WriteLine($"Зі стану {i} за T={horizonT} кроків:
{valueFunction[horizonT, i].ToString("F3", CultureInfo.InvariantCulture)}");
        }
    }

    public double SimulateOnce(int startState)
    {
        double totalSimulatedReward = 0;
        int currentState = startState;

        Console.Write($"Симуляція зі стану {startState}: [{currentState}");

        for (int t = 0; t < horizonT; t++)
        {
            int actionToTake = optimalPolicy[t, currentState];

            if (actionToTake == -1)
            {
                Console.Write($" -> !Дію не знайдено в стані {currentState} на кроці
{t}! ");
                break;
            }

            totalSimulatedReward += rewards[currentState, actionToTake];

            double randomRoll = randomGenerator.NextDouble();
            double cumulativeProb = 0.0;
```

```
            int nextState = -1;

            for (int j = 0; j < numStates; j++)
            {
                cumulativeProb += transitionProbs[currentState, actionToTake, j];
                if (randomRoll < cumulativeProb)
                {
                    nextState = j;
                    break;
                }
            }
            if (nextState == -1) {
                Console.Write($" -> !Помилка переходу з {currentState} дією
{actionToTake}!");
                nextState = currentState;
            }

            currentState = nextState;
            Console.Write($" -(дія {actionToTake})-> {currentState}");
        }
        Console.WriteLine($"] -> Загальна винагорода:
{totalSimulatedReward.ToString("F3", CultureInfo.InvariantCulture)}");
        return totalSimulatedReward;
    }
    public void RunSimulations(int numSimulationsPerStartState = 1)
    {
        if (horizonT == 0)
        {
            Console.WriteLine("\nНемає кроків для симуляції.");
            return;
```

```
        }
        Console.WriteLine($"\n--- Симуляція Процесу (по
{numSimulationsPerStartState} разів для кожного початкового стану) ---");
        for (int currentStartState = 0; currentStartState < numStates;
currentStartState++)
        {
            double sumOfRewardsForState = 0;
            Console.WriteLine($"\n--- Симуляції, що починаються зі стану:
{currentStartState} ---");
            for (int k = 0; k < numSimulationsPerStartState; k++)
            {
                sumOfRewardsForState += SimulateOnce(currentStartState);
            }
            Console.WriteLine($"Середня винагорода для стартового стану
{currentStartState} за {numSimulationsPerStartState} симуляцій:
{(sumOfRewardsForState / numSimulationsPerStartState).ToString("F3",
CultureInfo.InvariantCulture)}");
            Console.WriteLine($"Очікувана винагорода
(V*[{horizonT},{currentStartState}]): {valueFunction[horizonT,
currentStartState].ToString("F3", CultureInfo.InvariantCulture)}");
        }
    }


    public static void Main(string[] args)
{
    int nStates = 5;
    int mActions = 2;
    int T = 3;
```

```
double[,,] P = new double[nStates, mActions, nStates];
double[,] R = new double[nStates, mActions];

R[0, 0] = -1; R[0, 1] = -3;
R[1, 0] = -1; R[1, 1] = -3;
R[2, 0] = -1; R[2, 1] = -3;
R[3, 0] = 100; R[3, 1] = 100;
R[4, 0] = -50; R[4, 1] = -50;

//Ймовірності переходу P[стан, дія, наступний стан]
P[0, 0, 0] = 0.1; P[0, 0, 1] = 0.8; P[0, 0, 2] = 0.1; P[0, 0, 3] = 0.0; P[0, 0, 4] = 0.0;
P[0, 1, 0] = 0.0; P[0, 1, 1] = 0.5; P[0, 1, 2] = 0.3; P[0, 1, 3] = 0.2; P[0, 1, 4] = 0.0;
P[1, 0, 0] = 0.0; P[1, 0, 1] = 0.2; P[1, 0, 2] = 0.6; P[1, 0, 3] = 0.2; P[1, 0, 4] = 0.0;
P[1, 1, 0] = 0.0; P[1, 1, 1] = 0.1; P[1, 1, 2] = 0.3; P[1, 1, 3] = 0.5; P[1, 1, 4] = 0.1;
P[2, 0, 0] = 0.0; P[2, 0, 1] = 0.0; P[2, 0, 2] = 0.3; P[2, 0, 3] = 0.4; P[2, 0, 4] = 0.3;
P[2, 1, 0] = 0.0; P[2, 1, 1] = 0.0; P[2, 1, 2] = 0.1; P[2, 1, 3] = 0.3; P[2, 1, 4] = 0.6;
P[3, 0, 0] = 0.0; P[3, 0, 1] = 0.0; P[3, 0, 2] = 0.0; P[3, 0, 3] = 1.0; P[3, 0, 4] = 0.0;
P[3, 1, 0] = 0.0; P[3, 1, 1] = 0.0; P[3, 1, 2] = 0.0; P[3, 1, 3] = 1.0; P[3, 1, 4] = 0.0;
P[4, 0, 0] = 0.0; P[4, 0, 1] = 0.0; P[4, 0, 2] = 0.0; P[4, 0, 3] = 0.0; P[4, 0, 4] = 1.0;
P[4, 1, 0] = 0.0; P[4, 1, 1] = 0.0; P[4, 1, 2] = 0.0; P[4, 1, 3] = 0.0; P[4, 1, 4] = 1.0;
```

```
    FiniteHorizonMDPSolver solver = new FiniteHorizonMDPSolver(nStates,
mActions, T, P, R);

    solver.Solve();
    solver.PrintResults();
    solver.RunSimulations(numSimulationsPerStartState: 4);
}
}
```